The NetLogger Toolkit

for version 2.2.1, 1 October 2003

Data Intensive Distributed Computing group (http://www-didc.lbl.gov)
Contact: Dan Gunter (dkgunter@lbl.gov)

This manual is for the NetLogger Toolkit (version 2.2.1, 1 October 2003).

For further information about this notice, contact:

Brian L. Tierney Bld. 50B, Rm. 2239, Lawrence Berkeley National Laboratory, Berkeley, CA, 94720 email: bltierney@lbl.gov

Table of Contents

1 Getting NetLogger

Visit the NetLogger home page at http://www-didc.lbl.gov/NetLogger/ and click on the "Download" link under the "NetLogger components" menu. All of NetLogger except for the visualization tool NLV are at:

ftp://george.lbl.gov/pub/NetLogger/netlogger-2.2.1.tar.gz

After downloading the file, untar it with:

```
gzip -cd <file> | tar xvf -
```

or, if you're using GNU tar, the more compact:

```
tar xzvf <file>
```

Then, `cd` into the new directory and read the INSTALL and README text files for further instructions. Also please note the LICENSE file in the top-level directory, explaining the terms of use of the software.

2 Synchronizing system clocks

When using NetLogger to instrument a distributed application (including applications running on a cluster, of course), it is important that all the host clocks are synchronized as closely as possible.

Install and configure NTP `http://www-didc.lbl.gov/NetLogger/NTP.html` on hosts that are part of your distributed system. This will ensure that the clocks of all hosts are synchronized. This is essential for getting meaningful results from NetLogger.

3 NetLogger Events

Each log entry is called an *event*.

Each event is timestamped separately, either automatically or with a user timestamp. The following standard information is part of each log:

DATE The timestamp, to microsecond precision, in an ISO8601 format YYYYMMDD-Dhhmmss.<usec> For example: `20030911200754.234807`

HOST The DNS IP address of the hostname

PROG A user-supplied program name, or 'unknown' if none is given

NL.EVNT The event name, used as the primary means of identifying what "type" of event this is.

NL.ID An automatically generated string that is a universal unique identifier (UUID) for this particular NetLogger output instance.

LVL The logging level of this event. Each write of an event is associated with a log level. The default is zero (0). At any given time, NetLogger allows levels 0...N to be logged. In other words, when level N events are being written, then level 0...(N-1) events will also be written.

A "hello, world" for NetLogger (on a host not in DNS) looks like this:

```
DATE=20030911200754.234807 HOST=127.0.0.1 PROG=unknown \

NL.EVNT=hi NL.ID=jDAAABrWYD/UlAMA LVL=0
```

Before you go on, take aa look at the "best practices". These make it easier to use NetLogger tools and archives.

3.1 NetLogger Event Best Practices

NetLogger only requires a timestamp, hostname and program name in each event, and beyond that the names of the data elements are completely free-form.

In order to ease interoperability, I've come up with the following 'best practices'.

- If you have multiple variables to record, use one NetLogger event for each. This is particularly important if you want to use the NetLogger visualization tool, NLV.
- Use the following "well-known" fields

  'VAL' Events should have a field called VAL, whose value is the "main" value for that event.

  'ID' Events that want to be identified with (ie. grouped) with other lines should set the same ID for all lines (in the case below I used the process id for ID). The value may be a string or number.

'SRC' Dotted-decimal IP address of the source of measured resource (e.g. link source). If absent, it is assumed that the HOST is also the SRC. Therefore, all events in which the HOST and SRC are different must have a SRC field.

'DST' Dotted-decimal IP address of the destination of the measured resource (e.g., link destination). All network events should have a DST.

- Use descriptive, hierarchical event names

  For example, see the Global Grid Forum Discovery and Monitoring Event Description working group's http://www-didc.lbl.gov/damed/ names. Dots or underscores can separate parts of event names.

  It will be much harder to use and understand your log files later, if you name your events things like "Event1","Event2", etc.

4 Instrumentation

4.1 Language-independent Instrumentation Guidelines

The normal sequence of actions for instrumenting with NetLogger is:

1. Open the output url

   NetLogger can log to a local file, a TCP socket, or syslog. Many of NetLogger's parameters can be set with special 'query-string' additions to the output URL. See http://www-didc.lbl.gov/NetLogger/2.2/NetLogger_URL.html for more details.

2. Add NetLogger `write`s to key places in your source code. See the sample code for an example of how to do this. You should generate NetLogger events for all important events. See Chapter 3 [NetLogger Events], page 3.

   See Section 3.1 [Best practices], page 3, for naming conventions.

3. Close NetLogger before exiting (Python and Java do this automatically)

4.2 NetLogger Format Strings

In all languages, the NetLogger write tends to follow an old API set by the C `printf` family of functions, in which there is a string with special format codes, and then a list of values to be substituted for those codes.

However, the acceptable format string is not the same as the one allowed by `printf`. It is much more restricted.

Only the following format codes (or "typecodes") are allowed:

'`%f`' 32-bit floating point number

'`%lf`' 64-bit floating point number

'`%d`' 32-bit signed integer

'`%ld`' 64-bit signed integer

'`%s`' String of 255 characters or less

None of the precision or field length modifiers allowed by printf() are legal. Only one format code may be used per field. The following are WRONG:

```
"ip=%d.%d.%d.%d"    // more than one format code for 'ip'
"num=%.3f"          // precision modifiers not allowed
"dog=cat"           // right-hand side must be a %(typecode)
"95%%confidence=%f" // cannot use '%' as part of a field name
```

4.3 NetLogger Triggering

4.3.1 Quickstart

The NetLogger trigger mechanism allows one to tell running applications where to send their logging, even after they've started. Changing the log destination can be done using a text editor (e.g., vim) or through the NetLogger Activation Service.

Using just a text editor, here's how you do it:

.

1. Make the (original) destination for NetLogger be a "trigger URL" (see NetLogger URLs), and indicate a filename to use as a "trigger file". For example:

   ```
   setenv NETLOGGER_DEST x-nltrigger:/tmp/trigger-file
   ```

2. Start the application. NetLogger will initially write nowhere because the trigger file does not exist.
3. Edit the trigger file and add a "default" destination.

   ```
   # Default destination
   @url /tmp/default.log
   ```

   You should notice, after your next NetLogger `write`, that NetLogger switches to this log file.
4. Change the trigger file, for example to set the log level and destination for all programs named 'Foo':

   ```
   # Log level 2 and destination for foo
   PROG Foo
   @url x-netlog://remote.host?level=2

   # Default destination
   @url /tmp/default.log
   ```

4.3.2 Trigger File Format

A NetLogger trigger file is divided into sections, each separated by exactly one blank line. Within the sections, each line can fill one of three roles. Each of these lines has leading and trailing whitespace stripped. Then, if the line starts with the '#' character, it is a comment. If the line starts with the '`@`' character, it is a keyword. Otherwise, the line is a name/value pair where the first whitespace-delimited token is taken to be the 'name', and the rest is the 'value'. Each of these is described in more detail below:

- Comments are ignored. Note that there is no continuation character, so this:

  ```
  # comment \
  something else
  ```

  will parse the line "something else" as a name/value pair.
- Keywords indicate actions to take. Recognized keywords are:
  - '`@url`': The output URL to switch to. The @url keyword is required.
- Name/value pairs match constants in the NetLogger handle. These include all the standard NetLogger constants HOST, PROG, and NL.ID as well as any user-defined constants added programatically or through the URL query string.

The "name" in the name/value pair must be a simple string that exactly matches the constant name in the NetLogger handle. The "value" is a POSIX regular expression. Parentheses can be used for grouping and should not be backslash-escaped.

If the NetLogger output handle matches all the name/value pairs, then the actions specified by the keywords are taken. Like a programming language 'case' statement, the first match is used, so to specify a "default" url, simply put a one-line section at the end of the trigger file:

```
#end of previous section..

@url default-destination
<<EOF>>
```

4.3.3 Example

```
#
# NetLogger URL to send to:
#
@url /tmp/some-app-file.log?level=2
#
# Match based on these name/value pairs.
# Each has the format: Match-Name Match value
# The Match-name cannot have any spaces.
# The Match value may have spaces, but leading and trailing
# whitespace is stripped.
#
PROG some_(kind|sort)_of_app
DN   o=Grid u=dang

# Only one (1) blank line is allowed!!
# Next section starts here.
# No more blank lines are allowed until the end of the section.
@url /dev/null
PROG test
```

4.4 Instrumenting C/C++ source code

Some sample code is available at

http://www-didc.lbl.gov/NetLogger/sample_code.html

Automatically generated documentation is at

http://www-didc.lbl.gov/NetLogger/2.2.1/C_API/

Sorry, this section is not finished.

4.5 Instrumenting Python source code

Some sample code is available at http://www-didc.lbl.gov/NetLogger/sample_code.html.

Automatically generated documentation is at

http://www-didc.lbl.gov/NetLogger/2.2.1/Python_API/

4.5.1 Python NetLogger write API

The basic usage of NetLogger in Python is simple:

1. Import the module. The NetLogger module is nested rather deeply in its own namespace, so typically in Python one uses `import...as`:

   ```
   import gov.lbl.dsd.netlogger as netlogger
   ```

2. Create a 'Writer' with the `open()` function. The first argument to the function is the NetLogger URL. This function has a lot of keyword parameters. The most important ones are:

   `program` The name of your program, like "iperf" or "cool-app9"

   `reconnect`
   In the case of a TCP socket destination, this indicates how often in seconds to attempt to reconnect if the socket goes down. By default this is -1, meaning "don't!", do you *must* set this if you want NetLogger reliability. A good typical value is 30 seconds. Note that you can also set this by adding `reconn=<num>` to the NetLogger URL.

   ```
   # Set all parameters programmatically
   writer = netlogger.open("x-netlog://remote.host",
                           program="sample",
                           reconnect=30)
   # Set reconnect in the URL
   writer = netlogger.open("x-netlog://remote.host?reconn=30",
                           program="sample")
   ```

3. Use the `write()` method on the writer object. The first argument is the log level, the second one is the Chapter 3 [NetLogger event name], page 3, the third argument is the Section 4.2 [format string], page 5, and then after that a list of values that match the format string. For example:

   ```
   # Write a message
   nl.write(0, "Sample.start",
            "VAL=%d METHOD=%s PID=%d",
            12, 'simple', os.getpid())
   ```

4. When the Writer object created with `open()` goes out of scope, the NetLogger output will clean itself up and close. Of course, you can hasten this process by simply deleting it:

   ```
   del writer
   ```

4.5.2 Python NetLogger read API

Sorry, this section is not finished.

4.6 Instrumenting Java using the Java wrapper to the C library

Automatically generated documentation is available at:
http://www-didc.lbl.gov/NetLogger/2.2.1/Java_Wrapper_API/

4.7 Instrumenting Java using log4j

The *pure* Java API is currently not functional. We are looking into using a log4j appender for a "pure" Java API, as well as reviving the Java-over-C if anyone ever needs it.

5 NetLogger command-line tools

Note: The documentation on these tools is incomplete. Those tools that *are* really documented are marked with a '!'

5.1 netlogd: TCP socket server daemon

Netlogd accepts one or more NetLogger TCP streams and writes them to one or more NetLogger output URL's. Typically, just one local file URL is used for output. Optionally, it can report its aggregate throughput.

USAGE netlogd.py [-dfhopqtu]

OPTIONS

'`-d <num>`'	Set the debug level, 0=none 10=all Default: 0
'`-f <fmt>`'	The format for output messages. May be 'ulm' or 'bin[ary]' Default: ulm
'`-h`'	Print this message
'`-o <url>`'	The output URL for NetLogger messages, repeatable
'`-p <portnum>`'	The port to listen for incoming messages Default: 14830
'`-q`'	Set quiet mode, no output; overrides debug level
'`-t <num>`'	Report aggregate throughput approx. every 'num' messages
'`-u <url>`'	NetLogger debug log URL (implies -t)

5.2 Socket-server, save logs to MySQL

Sorry, not documented yet.

5.3 Summarize a ULM log file

Sorry, not documented yet.

5.4 Linux CPU sensor

Sorry, not documented yet.

5.5 nlforward: Log file forwarder

This tool forwards a single NetLogger file or directory of files to an output URL.

If the input dir-or-file is a file, then that file will be forwarded (with all reconnection logic as below), and then the program will exit.

If the input dir-or-file is a directory, then all files in that directory will be forwarded, and the directory will be rescanned every N seconds. The program will run until killed. Any file which has been determined to be open for writing by another process will not be forwarded. This avoids the problem of forwarding and deleting files that are still growing.

This tool is designed to be robust and persistent in its attempts to forward files. Failure to forward a file is not fatal to the program, it will just move on to the next one. If a file is successfully forwarded, it is deleted. Failure to forward a file will result in its being left alone. If the **output** URL fails, nlforward will periodically "re-try". The environment variable `NETLOGGER_DEST`, if set, will override the output URL.

Several features are present to avoid forwarding – and deleting! – an active log file. If the log file is being written using NetLogger write APIs, then advisory file locking will prevent an open file from being forwarded. If the '`-m`' option is given, no file that is younger than the specified time will be forwarded. Finally, if the '`-l`' option is given, `lsof` is used during every directory scan to see which files are currently open for writing by another process (obviously, systems without lsof do not do this). This can compensate for logs which are created by other systems that do not use advisory file locking.

USAGE nlforward.py [-h-Lmnqrs] <input dir-or-file> <output URL>

OPTIONS

'`-h`'	Print this message
'`-L`'	Do not use 'lsof', even if it is available
'`-m sec`'	A file can only be forwarded if its modification time is this number of seconds in the past (default=0,don't check)
'`-n url`'	Log a copy of output messages to NetLogger at '`url`'
'`-q`'	Quiet mode, print nothing to screen; NetLogger logging will still be done, if '-n' option has been given
'`-r <sec>`'	Reconnect interval in seconds Default: 30
'`-s <sec>`'	Re-scan directory interval in seconds. Default: 60. Must be an integer >= 1.

EXAMPLES

```
# Forward all files in /tmp/netlogger to 'remote.host',
# re-scanning the directory every 10 minutes
nlforward.py -s 600 /tmp/netlogger x-netlog://remote.host

# Forward 'foo.log' to 'remote.host', turning off the
```

```
# reconnect behavior
nlforward.py -r '-1' foo.log x-netlog://remote.host
```

5.6 Ganglia wrapper

Sorry, not documented yet.

5.7 Linux memory usage sensor

Sorry, not documented yet.

5.8 Send fake log data at different rates

Sorry, not documented yet.

6 Further reference

1. Summary paper on NetLogger

Index